FIFO

Last updated 10/1/26

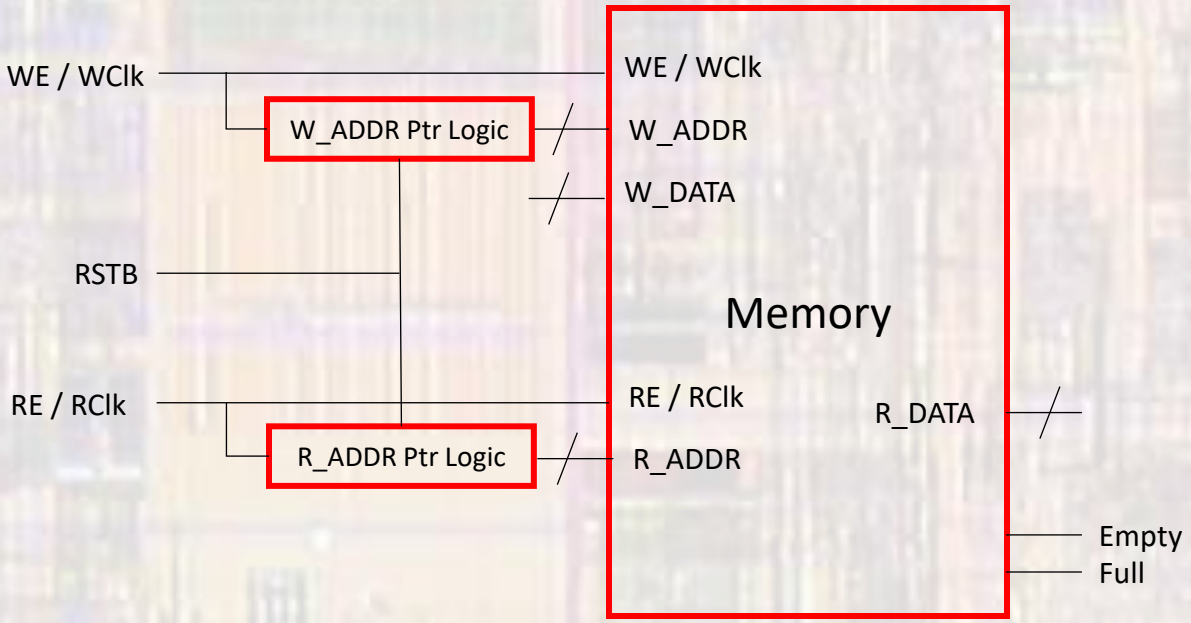
FIFO

- First In First Out Memory
  - Typically used to buffer data that is written and read at different rates or sporadically
    - Clock Domain Crossing
      - Safely transfer data between subsystems running on independent or asynchronous clock frequencies
    - Rate Matching / Speed Buffering
      - Temporarily store data when a producer device generates data faster than a consumer device can process it, or vice versa
    - Burst Absorption
      - Absorb sudden bursts of incoming data so the downstream processing logic does not drop data while getting ready – communications systems
    - Communication Interfaces & Peripherals
      - They are integrated into serial and parallel communication controllers—like UART, SPI, and I2C—to hold incoming or outgoing bytes until the main system or processor gets to them.
    - Data Streaming & Signal Processing
      - Used in digital signal processors (DSPs) and audio/video pipelines to maintain strict chronological order during real-time data streaming.
    - Off-Chip Memory Buffering
      - They queue and align data blocks before bursting them out to external memory components like DRAM or SRAM

© tj

FIFO

- Simplified Block Diagram



© tj

FIFO

- Simplified Operation

Action	W_PTR	R_PTR	Data In
Reset	0	0	x
W	0 → 1	0	A
W	1 → 2	0	B
W	2 → 3	0	C
W	3 → 0	0	D
R	0	0 → 1	x
R	0	1 → 2	x
W	0 → 1	2	E
R	1	2 → 3	x
R	1	3 → 0	x
R	1	0 → 1	x

ADDR	DATA
0	A / E
1	B
2	C
3	D

CNT	Full	Empty	Data Out
0	0	1	x
1	0	0	x
2	0	0	x
3	0	0	x
4	1	0	x
3	0	0	A
2	0	0	B
3	0	0	x
2	0	0	C
1	0	0	D
0	0	1	E

© tj

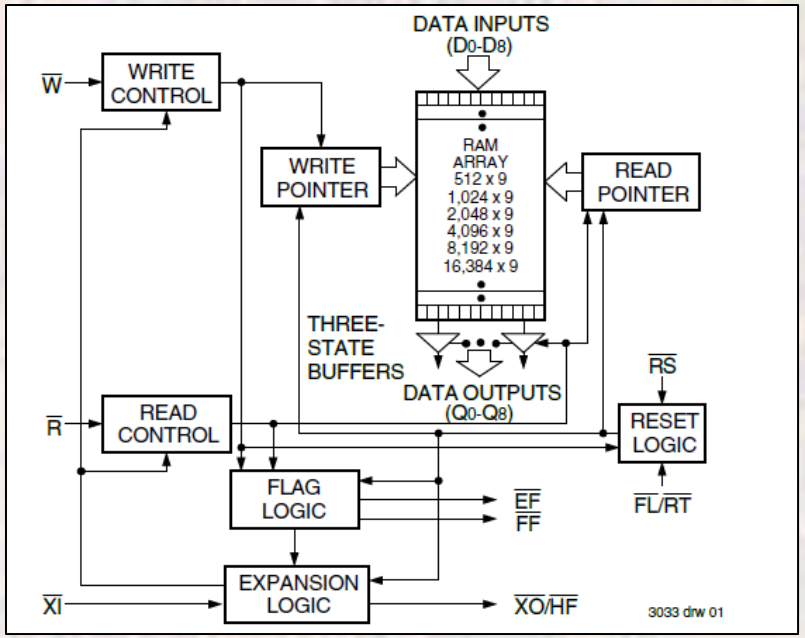
FIFO

- Asynchronous Example

RENESAS

3.3 VOLT CMOS ASYNCHRONOUS FIFO
512 x 9, 1,024 x 9,
2,048 x 9, 4,096 x 9,
8,192 x 9, 16,384 x 9

IDT72V01, IDT72V02
IDT72V03, IDT72V04
IDT72V05, IDT72V06



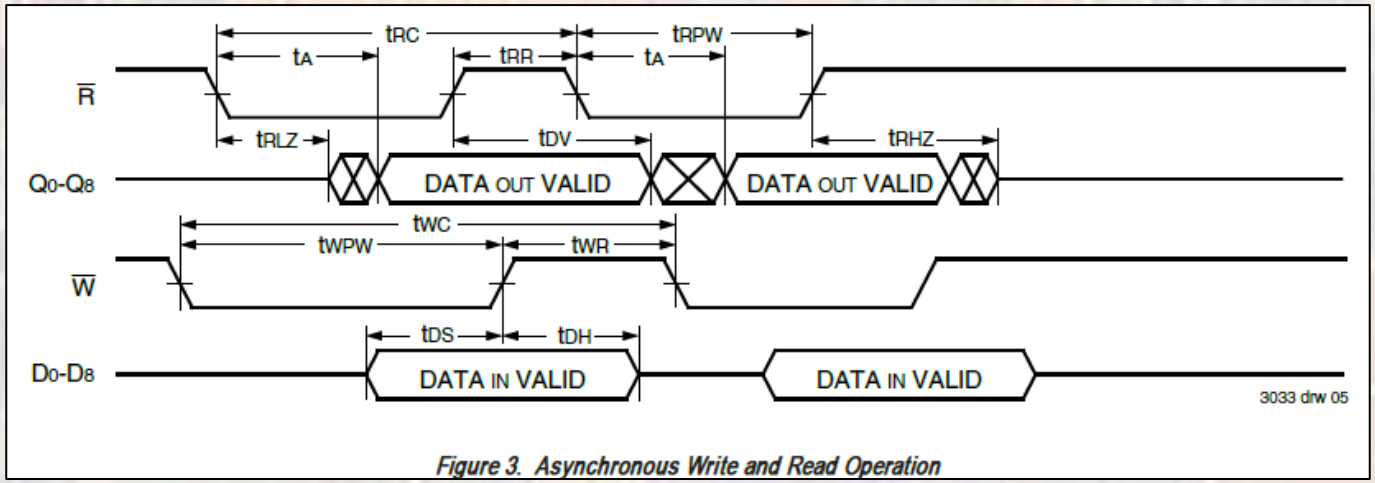


Figure 3. Asynchronous Write and Read Operation

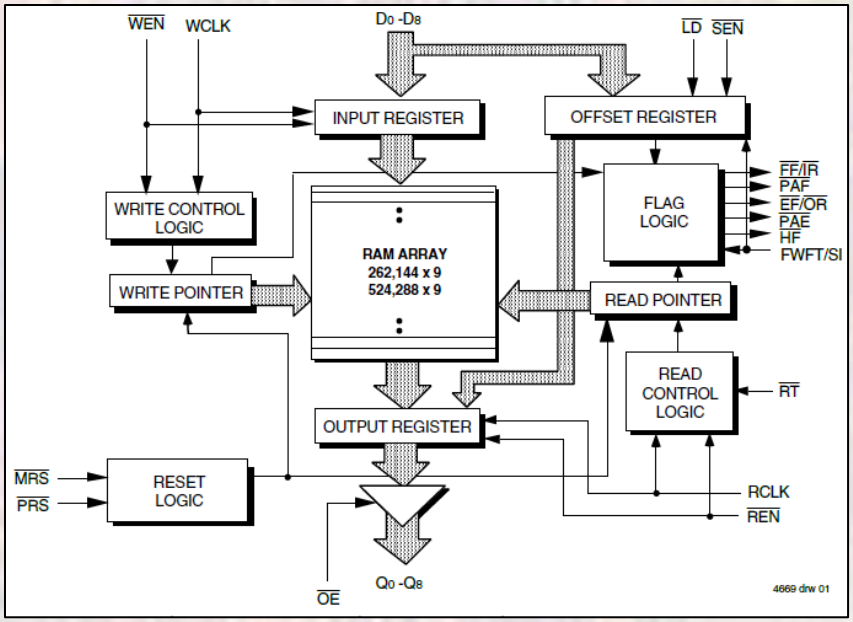
© tj

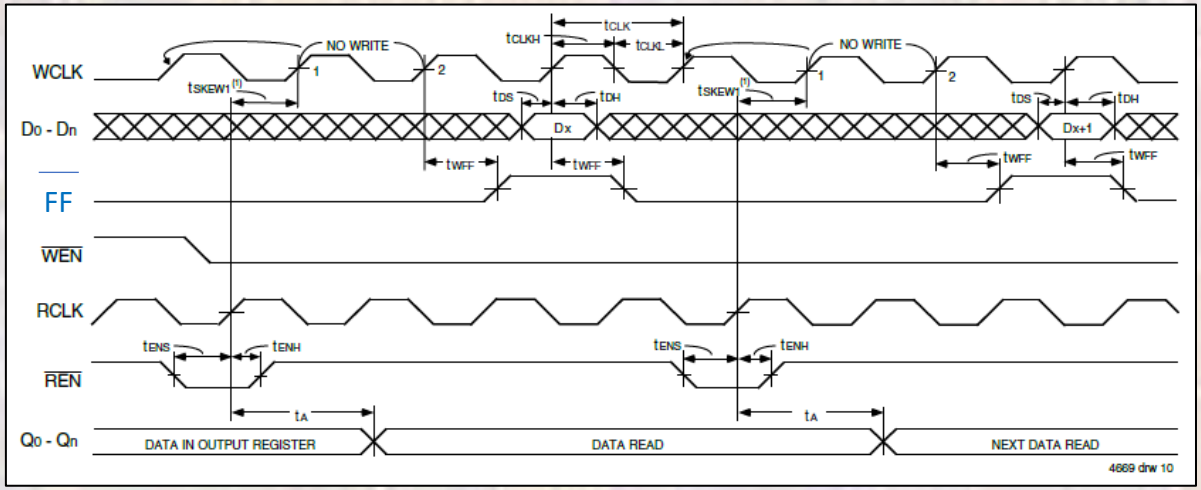
FIFO

- Synchronous Example

RENESAS

3.3 VOLT HIGH DENSITY CMOS SUPERSYNC FIFO™
262,144 x 9 — IDT72V2101
524,288 x 9 — IDT72V2111





© tj

FIFO

- VHDL – Entity / signals

```
----------------------------------------
--
-- FIFO_generic.vhdl
--
-- created 9/20/26
-- tj
--
-- rev 0
----------------------------------------
--
-- FIFO example for class notes
--
----------------------------------------
--
-- Inputs:  clk, rstb, wr_en, rd_en, data_in
-- Outputs: full, empty, rw_error, data_out
--
-- NOTE: No test for non-valid addresses if FIFO_DEPTH not a power of 2
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
```

```
entity FIFO_generic is
   generic(
         FIFO_DEPTH : natural := 16;
         WORD_SIZE  : natural := 8
         );
   port(
         i_clk:      in  std_logic;
         i_rstb:     in  std_logic;
         i_wr_en:    in  std_logic;
         i_rd_en:    in  std_logic;
         i_data_in:  in  std_logic_vector((WORD_SIZE - 1) downto 0);

         o_data_out: out std_logic_vector((WORD_SIZE - 1) downto 0);
         o_full:     out std_logic;
         o_empty:    out std_logic;
         o_rw_error: out std_logic
      );
end entity FIFO_generic;
```

```
architecture behavioral of FIFO_generic is

   constant ADDR_WIDTH:    natural := integer(ceil(log2(real(FIFO_DEPTH))));
   constant COUNT_WIDTH:   natural := integer(ceil(log2(real(FIFO_DEPTH + 1))));

   -- fifo - a memory array
   type fifo_mem_type is array(0 to (FIFO_DEPTH - 1)) of std_logic_vector((WORD_SIZE - 1) downto 0);
   signal fifo: fifo_mem_type;

   -- w_ptr  points to the location where the next word will be written
   -- r_ptr  points to the location where the next word will be read from
   -- fifo_cnt - current number of valid entries
   signal w_ptr:     unsigned((ADDR_WIDTH - 1) downto 0);
   signal r_ptr:     unsigned((ADDR_WIDTH - 1) downto 0);
   signal fifo_cnt:  unsigned((COUNT_WIDTH - 1) downto 0);

   -- pointer control signals, not required but make the code more readable
   signal wr_allowed:   std_logic;
   signal rd_allowed:   std_logic;

begin
```

FIFO

- VHDL – w/r control

```
-- r/w control process
process(o_full, o_empty, i_rd_en, i_wr_en, i_clk, i_rstb)
begin
   wr_allowed <= i_wr_en and not o_full;
   rd_allowed <= i_rd_en and not o_empty;

   if(i_rstb = '0') then
      o_rw_error <= '0';
   elsif(rising_edge(i_clk)) then
      o_rw_error <= (i_wr_en and o_full) or (i_rd_en and o_empty);
   end if;
end process;
```


© tj

FIFO

- VHDL – w/r pointers

```
-- write pointer
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      w_ptr <= (others => '0'); -- top of fifo

   elsif(rising_edge(i_clk)) then
      if(wr_allowed = '1') then
         if(w_ptr = (FIFO_DEPTH - 1)) then
            w_ptr <= (others => '0');  -- wrap the fifo
         else
            w_ptr <= w_ptr + 1;
         end if;
      end if;
   end if;
end process;
```

```
-- read pointer
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      r_ptr <= (others => '0'); -- top of fifo

   elsif(rising_edge(i_clk)) then
      if(rd_allowed = '1') then
         if(r_ptr = (FIFO_DEPTH - 1)) then
            r_ptr <= (others => '0');  -- wrap the fifo
         else
            r_ptr <= r_ptr + 1;
         end if;
      end if;
   end if;
end process;
```


© tj

FIFO

- VHDL – write/read

```vhdl
-- write
process(i_clk)
begin

   -- no reset condition

   if(rising_edge(i_clk)) then
      if(wr_allowed = '1') then
         fifo(to_integer(w_ptr)) <= i_data_in;
      end if;
   end if;
end process;
```

```vhdl
-- read
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      o_data_out <= (others => '0'); -- predictable output in reset

   elsif(rising_edge(i_clk)) then
      if(rd_allowed = '1') then
         o_data_out <= fifo(to_integer(r_ptr));
      end if;
   end if;
end process;
```


© tj

FIFO

- VHDL – entry tracking

```
-- count tracking
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      fifo_cnt <= (others => '0'); -- clear the count

   elsif(rising_edge(i_clk)) then
      if((wr_allowed = '1') and (rd_allowed = '0')) then
         fifo_cnt <= fifo_cnt + 1;
      elsif((wr_allowed = '0') and (rd_allowed = '1')) then
         fifo_cnt <= fifo_cnt - 1;
      else
         -- no change if rd and wr, or neither
         fifo_cnt <= fifo_cnt;
      end if;
   end if;
end process;
```


© tj

FIFO

- VHDL - status

```
   -- status full/empty
   process(fifo_cnt)
   begin
      if(fifo_cnt = 0) then
         o_empty <= '1';
      else
         o_empty <= '0';
      end if;

      if(fifo_cnt = FIFO_DEPTH) then
         o_full <= '1';
      else
         o_full <= '0';
      end if;
   end process;

end architecture;
```


© tj

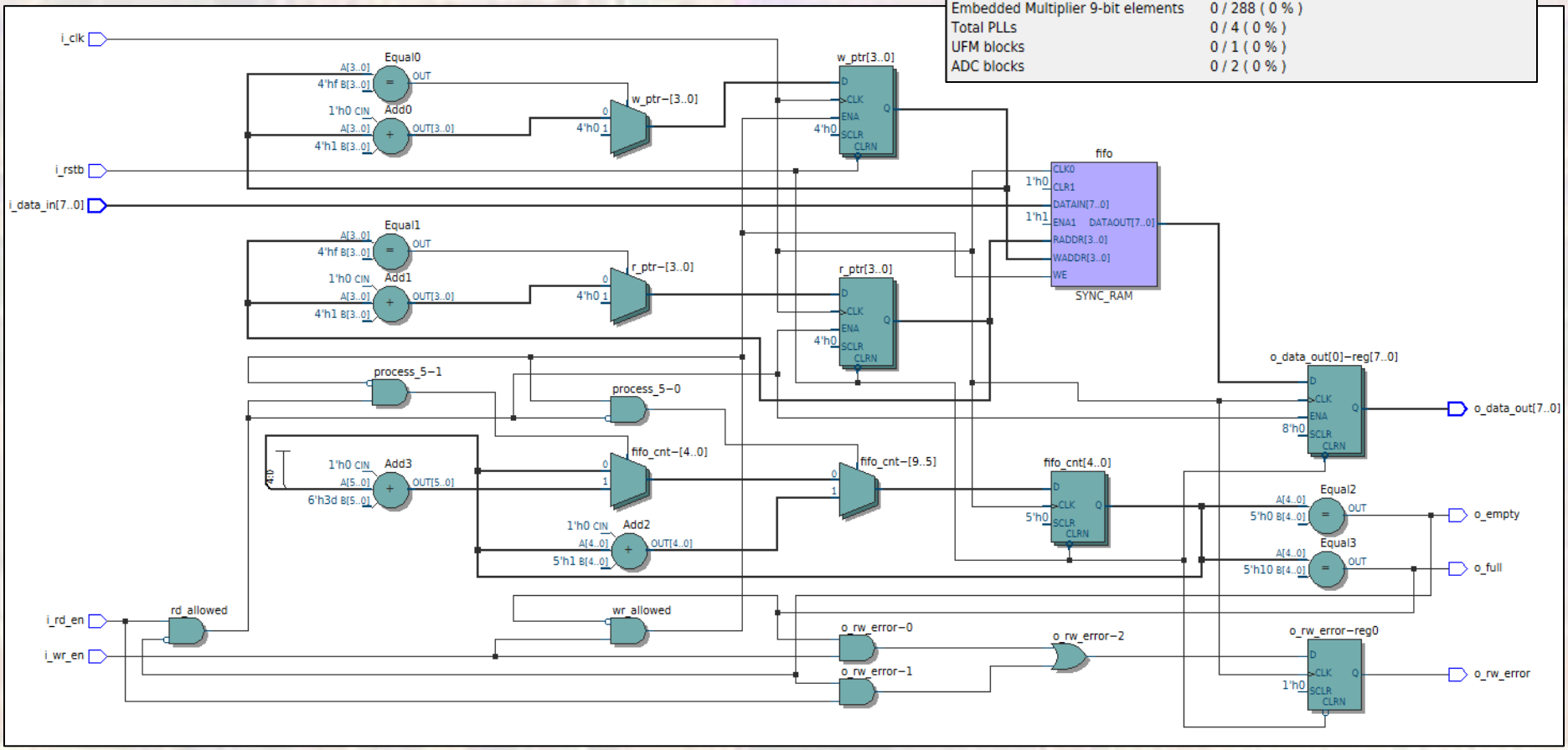
FIFO

- VHDL - RTL

Flow Summary

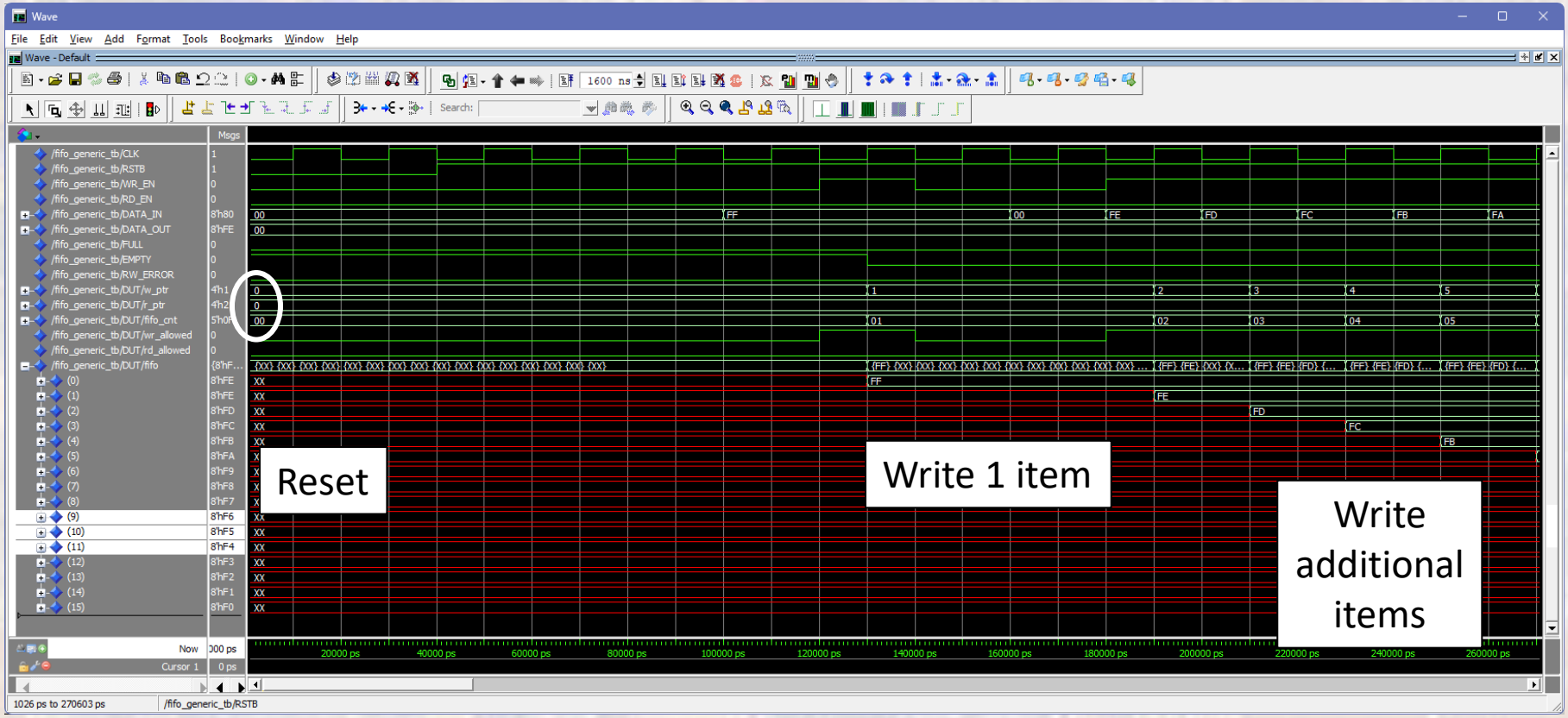
<<Filter>>

Flow Status	Successful - Thu Oct 1 14:03:12 2026
Quartus Prime Version	25.1std.0 Build 1129 10/21/2025 SC Lite Edition
Revision Name	fifo
Top-level Entity Name	FIFO_generic
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	60 / 49,760 (< 1 %)
Total registers	48
Total pins	23 / 360 (6 %)
Total virtual pins	0
Total memory bits	128 / 1,677,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)



FIFO

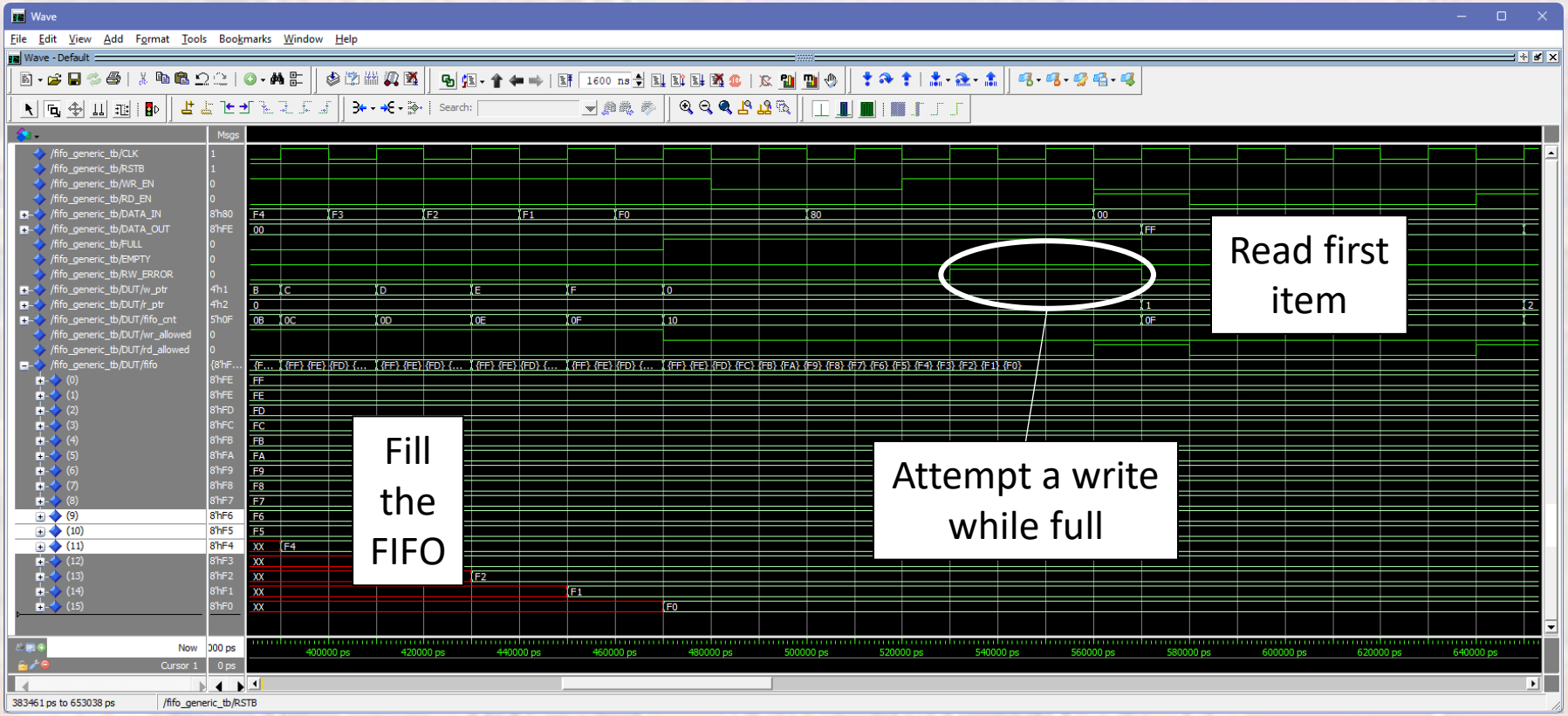
- VHDL - SIM



 © tj

FIFO

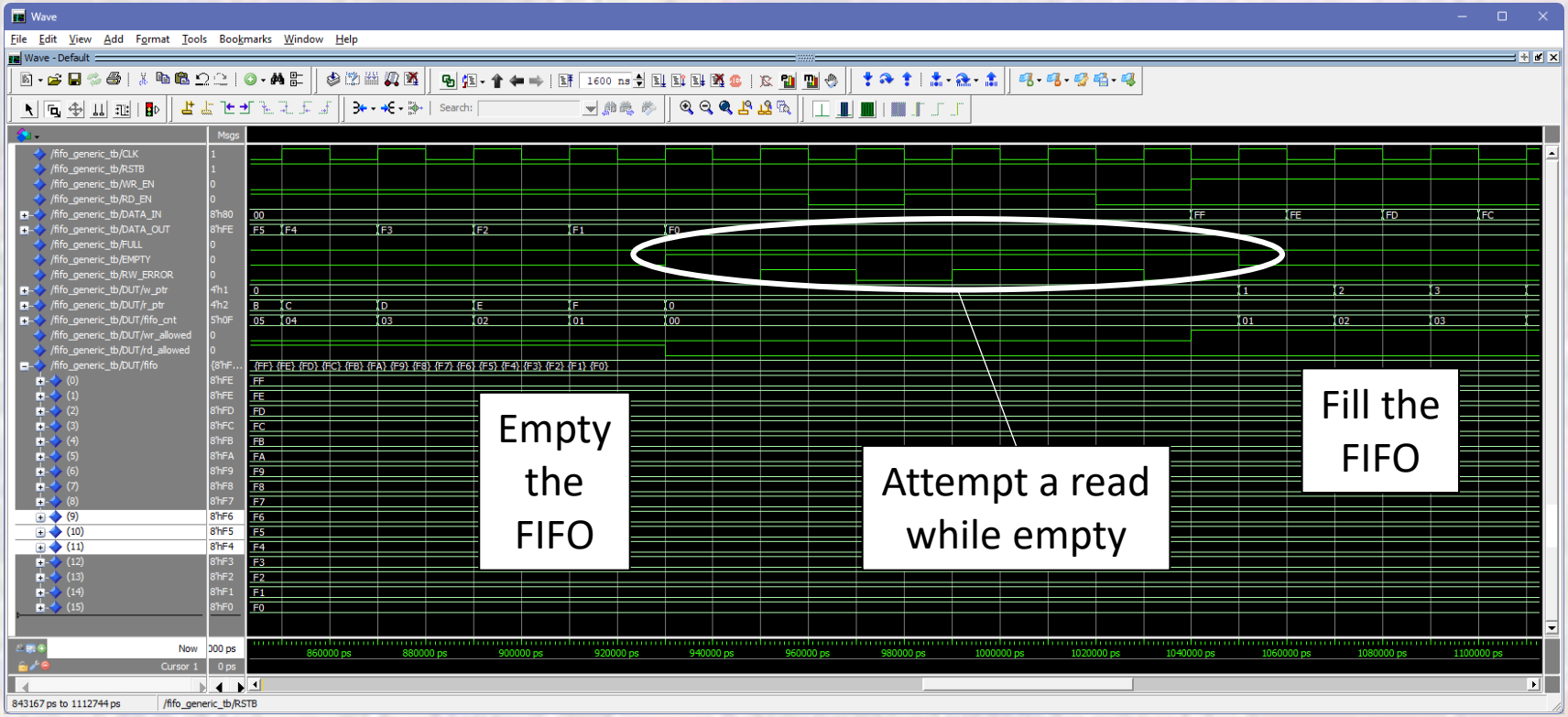
- VHDL - SIM



 © tj

FIFO

- VHDL - SIM



 © tj

FIFO

- VHDL - SIM

